Proyecto final
Métodos avanzados en aprendizaje automático

Alejandro Bellogín Kouki
Universidad Autónoma de Madrid
alejandro.bellogin@uam.es

19 de mayo de 2008

1. Introducción

En este proyecto se planteó la resolución de un problema real: calcular los individuos que con mayor probabilidad se fugarían de una empresa de seguros de coches. El problema propuesto obliga a manejar al menos dos dificultades:

- Clases desbalanceadas, siendo la minoritaria la que se quiere predecir.
- Elevado número de atributos, con el consiguiente tiempo de preprocesamiento.

En este trabajo, se explicará cómo se ha abordado el problema, analizando cada una de las fases por las que se ha pasado, sus técnicas, alternativas estudiadas y elecciones tomadas. Se mencionarán las herramientas utilizadas para llevar todo esto cabo, examinando también las distintas posibilidades encontradas y las razones por las que se eligió como algoritmo de clasificación *bagging con* $M5'$.

2. Métodos utilizados

En esta sección se describirán con detalle los pasos llevados a cabo. En todo proyecto de aprendizaje automático se incluyen las siguientes fases [4]:

- Recolección de los datos
- Elección de los atributos
- Elección del modelo
- Entrenamiento del clasificador
- Evaluación del clasificador

En este caso particular, las dos primeras fases vinieron en parte dadas, ya que no se tuvo que extraer los datos de ninguna base de datos, ni pensar los atributos a incluir de partida en dicho conjunto. Además, no hubo capacidad de elección para decidir el número de ejemplos que se necesitaban para garantizar un buen comportamiento del sistema, salvo por el hecho de la elección (en fases posteriores) del conjunto de entrenamiento y test.

Núm. ejemplos	Núm. atributos	Núm. total *missing values*	Atributo con más m. v.	Núm. atributos constantes
79999	70	54604	YCIA_ANT (17754)	2

Cuadro 1: Estadísticas del conjunto inicial

Núm. ejemplos	Núm. atributos
72702	42

Cuadro 2: Características del conjunto final de datos

Nuestra tarea, no obstante, comenzó justo en la segunda fase: elección de atributos. Como se verá a continuación, se realizó una selección de los atributos de acuerdo a distintos criterios, lo cual permitió disminuir el número de variables a estudiar en el problema.

Posteriormente, se pasó por cada una de las fases siguientes a la ya mencionada de manera iterativa, probando con distintos clasificadores, distintos parámetros para el mismo clasificador y hasta distintos conjuntos de datos para el mismo clasificador y conjunto de parámetros.

A continuación, se describen más en profundidad cada una de estas fases.

2.1. Preprocesamiento

La tabla 1 resume algunas estadísticas sobre el conjunto de datos inicial. Dado el gran número de parámetros es aconsejable para muchos algoritmos de clasificación (y para muchos otros es obligado) reducir el número de atributos.

La elección de los atributos se puede realizar de muy diversas formas:

- Inspección manual: se analiza *semánticamente* si un atributo será relevante al problema. Si se dispone de un experto en el área se obtienen buenos resultados, en otro caso, se necesitará mucho tiempo sin garantía de resultados satisfactorios. De esta manera se pueden crear nuevas variables a partir de otras ya existentes (ratios, diferencias, ...).
- Análisis estadístico: se pueden estudiar las correlaciones con la clase de los atributos, o aplicar un test de hipótesis a los atributos para averiguar si son estadísticamente significativos o no.
- Algoritmos de selección de atributos: existen algoritmos específicos (PCA, algoritmos genéticos, ranking, ...) que realizan una búsqueda en el espacio de atributos para elegir el menor subconjunto que permite predecir la clase de la mejor manera posible. Esta opción tiene un problema grave: el sobreajuste.
- Algoritmos de clasificación: se pueden utilizar para descubrir cuáles son los atributos que más discriminan (muy intuitivo en árboles de decisión)

A pesar de que los algoritmos de aprendizaje automático en su mayoría están diseñados para aprender cuáles son los atributos más apropiados para tomar las decisiones, si en la práctica se añaden atributos irrelevantes o redundantes al conjunto de datos, se provoca un deterioro del rendimiento [7]. Por ello, en este proyecto se utilizaron algunas de las técnicas mencionadas, obteniendo finalmente una base de datos con las características que se incluyen en la tabla 2.

La fase de preprocesamiento de datos se dividió en varias partes:

1. Selección de atributos. En esta fase se utilizaron algunas de las técnicas previamente mencionadas. Por una parte se analizaron los atributos junto con su descripción para detectar cuáles podrían ser inútiles para el clasificador (uno obvio era el identificador de la instancia). Se utilizó la matriz de covarianza y las correlaciones para detectar qué atributos eran constantes en los datos, además se almacenaron los que tuvieran correlaciones altas para comprobar en consiguientes transformaciones que dichos atributos no fueran eliminados. También se utilizaron tanto árboles de decisión como algoritmos de búsqueda de atributos (greedy, algoritmos genéticos) y los atributos que se encontraron más relevantes fueron los siguientes: ANT_MUT, ANT_POL, FP, TIPO, BONF, recibosdev. Cuando se había reducido el número de atributos a 42 se probó PCA y se obtuvo un subconjunto de 30 atributos (clase incluída), no obstante, en las primeras pruebas con los clasificadores no se produjeron buenos resultados. En el anexo A se puede ver la lista de los 42 atributos finalmente seleccionados.

2. Reemplazar los *missing values*. Los *missing values* se reemplazaron en todos los casos por la media del atributo. Se barajó la posibilidad de reemplazar la media según la clase de la instancia, pero se rechazó la idea al ser esto imposible con los datos de explotación.

3. Centrar. De esta manera cada atributo tiene media 0.

4. Normalizar. La desviación estándar de cada atributo vale 1.

5. Eliminar *outliers*. Detectamos *outliers* utilizando la siguiente fórmula basada en el identificador de Hampel [3]: si el valor de un elemento x para el atributo i cumple

$$x_i - \mu_i > 3{,}8 \cdot \sigma_i$$

   lo contabilizamos como *outlier* (donde μ_i y σ_i son la media y desviación típica del atributo i). Debido a que eliminamos todos los ejemplos que contengan algún *outlier* en alguno de sus parámetros, elegimos el parámetro que multiplica a la desviación estándar (en este caso, $3{,}8$, el valor estándar suele ser un poco menor) suficientemente grande para no quedarnos sin demasiados ejemplos. Además, la fórmula incluye la media y la desviación de cada atributo para que sea más general y se pueda aplicar aunque los atributos no estuvieran centrados y/o normalizados.

2.2. Entrenamiento y validación

Una vez se ha encontrado un subconjunto de los atributos iniciales que tenga un tamaño razonable y siga manteniendo suficiente información de la clase objetivo, llega el momento de entrenar y probar los modelos de clasificación. Para ello, un paso previo a este entrenamiento y prueba es la construcción de los conjuntos de datos que se van a utilizar en estas dos fases.

El problema al que nos enfrentamos en este proyecto es bien conocido en el área del Aprendizaje Automático: se tiene un conjunto de datos desequilibrado y se quiere realizar una predicción (o clasificación) sobre la clase minoritaria. En [2] encontramos otros ejemplos reales de esta situación: detección de casos fraudulentos en transacciones con tarjetas de crédito y llamadas telefónicas, diagnósticos médicos de enfermedades poco usuales, reconocimiento de derramamientos de aceite en imágenes por satélite de la superficie del mar o categorización de textos.

Una de las prácticas más habituales a la hora de enfrentarse a este problema es equilibrar las muestras de entrenamiento de cada clase. Otra opción es utilizar las Máquinas de Vectores Soporte (del inglés *Support Vector Machines*, SVM) [1] las cuales son robustas incluso en situaciones que incluyen datos etiquetados y sin etiquetar (la clase es desconocida) [7]. En este trabajo se utilizó la primera opción.

Existen dos maneras básicas de equilibrar las muestras de entrenamiento: sobremuestreo de la clase minoritaria (repetir ejemplos) y submuestreo de la clase mayoritaria (eliminar ejemplos). En este trabajo se decidió realizar la segunda de las propuestas para entrenar el clasificador, mientras que para probarlo

se utilizó una muestra con la misma proporción de clases que en el conjunto inicial. Esta decisión se tomó planteando la hipótesis de que sería muy probable que el conjunto de explotación tuviera la misma distribución que el conjunto de construcción, no obstante, según [6] la distribución de clases original no siempre es la mejor para entrenar los algoritmos de clasificación, y esta distribución debería contener entre un $50\,\%$ y un $90\,\%$ de ejemplos minoritarios; por tanto, aquí se encuentra un punto no considerado en este trabajo y que sería adecuado confirmar en un futuro: comprobar si el hecho de utilizar un conjunto de entrenamiento y test no balanceado y con un mayor número de clases minoritarias mejora o no la predicción de la clase minoritaria.

Hay que mencionar que al principio del trabajo, se pensó utilizar un conjunto de validación para controlar el sobreajuste al entrenar, sin embargo, esto añadía más problemas a la consecución del objetivo final:

- ¿El conjunto de validación debía seguir una distribución igual que la del conjunto de entrenamiento o igual que la del conjunto de prueba?
- ¿Se debía utilizar una medida estándar para parar el entrenamiento (MSE) o la que luego aplicaríamos con test (orientada a campaña)?
- En cualquier caso, utilizar un conjunto de validación disminuiría el número de ejemplos de fuga disponibles para entrenar, por lo que los algoritmos tendrían menos casos de la clase que se pretendía predecir.

Por estas razones se decidió no validar los modelos y pasar a test después de haber entrenado.

A continuación se puede ver la manera en la que se ha creado el conjunto de entrenamiento, y en el apéndice B el código en Matlab equivalente.

Pseudocódigo utilizado para crear el conjunto de entrenamiento

```
Sea D el conjunto de datos preprocesado.
Sea D' un conjunto vacío.
Sea F el conjunto de datos de fuga y f su tamaño.
Se elige un subconjunto de F para entrenar,
        se llama F' y f' su tamaño.
Se añade el conjunto F' a D'.
Se eligen aleatoriamente f' elementos del conjunto D − F,
        se añaden a D'.
Se mezclan los elementos de D'.
En D' se obtiene un conjunto de datos balanceado.
```

De esta manera se crea un conjunto de entrenamiento balanceado, y en cada ejecución el conjunto de fuga y no fuga será distinto. El carácter aleatorio de este procedimiento se podría mejorar utilizando técnicas de equilibrado de clases basadas en patrones frontera [2], en estas técnicas no se utilizan los patrones que seguro se clasifican bien (no aportan información relevante) ni aquellos que no podrían ser clasificados correctamente bajo ningún discriminante (ruido).

2.3. Prueba

El objetivo de este proyecto consiste en conseguir el $10\,\%$ de los individuos que con mayor probabilidad pertenecen a la clase fuga. La prueba se realizó con este objetivo en mente, es decir, el procedimiento seguido fue (puede verse el código equivalente en el apéndice C):

1. A partir de un clasificador entrenado, se clasifican todas las instancias de test. Es necesario que el clasificador dé un valor (probabilidad de pertenencia) a cada instancia indicando el grado de que dicha instancia sea de clase fuga.

2. Se almacenan todos estos valores junto con el identificador de la instancia y se ordenan de mayor probabilidad a menor.

3. Elegimos el $10\,\%$ de instancias con valores más altos y comparamos su clase real: si es fuga, ha habido acierto, si no, fallo.

4. La precisión del clasificador se mide como el que más aciertos haya tenido.

Como ya se ha comentado anteriormente, se decidió tomar como conjunto de prueba uno en el que la proporción de clases fuera igual que la del conjunto inicial de datos (construcción). Este conjunto se construyó de la siguiente forma (el código Matlab aparece en el apéndice B):

Pseudocódigo utilizado para crear el conjunto de prueba

```
Sea D el conjunto de datos preprocesado.
Sea pf la proporción de fuga en D.
Sea D'' un conjunto vacío.
Sea F el conjunto de datos de fuga y f su tamaño.
Se elige un subconjunto de F para test,
        se llama F'' y f'' su tamaño.
Se añade el conjunto F'' a D''.
Se eligen aleatoriamente f''*pf elementos del conjunto D - F,
        se añaden a D''.
Se mezclan los elementos de D''.
En D'' se obtiene un conjunto de datos desbalanceado.
```

Un aspecto interesante que se extrae de los pseudocódigos presentados para crear los conjuntos de datos (entrenamiento y test) es la elección del subconjunto de fuga para entrenamiento, y, su complementario, para test. En la mayoría de pruebas se ha utilizado una proporción de $0{,}4$ para test y $0{,}6$ para entrenamiento, con el objetivo de que el clasificador esté suficientemente bien entrenado. No obstante, también se han realizado pruebas con otras proporciones, como se verá en la siguiente sección.

2.4. Comparación de clasificadores

Una vez se han obtenido los datos de entrenamiento y test, llega el momento de la clasificación propiamente dicha. En este trabajo se han utilizado los siguientes clasificadores (para más detalles consultar [7]):

- Clasificador probabilístico estándar Naïve Bayes
- Regresión lineal estándar por mínimos cuadrados, puede realizar una selección de atributos o utilizar el conjunto entero de atributos e ir descartando uno a uno hasta llegar a una condición de parada
- Árbol de decisión $C4{,}5$ (en particular, la revisión número 8 o árbol de decisión `J48`)
- Árbol modelo $M5'$ (`M5P`), los árboles modelo almacenan en cada hoja un modelo de regresión lineal que predice el valor de clase de las instancias que llegan a dicha hoja (en contraste con los árboles de decisión, que almacenan en la hoja el valor de la clase que representa el valor medio de las instancias que llegan a dicha hoja)

- Árbol de decisión o de regresión que utiliza poda por reducción de error (`REPTree`) y reducción por ganancia de información
- Bosque de árboles aleatorios (consideran un número arbitrario de características en cada nodo, utiliza bagging)
- Mezcla de clasificadores mediante bagging
- Reglas conjuntivas, este clasificador aprende una sola regla para predecir la clase y asigna la clase por defecto a las instancias no cubiertas por dicha regla
- Obtención de reglas a partir de árboles de decisión parciales usando $J4{,}8$ (`Part`)
- Perceptrón multicapa, red neuronal entrenada mediante retropropagación
- Mezcla de clasificadores mediante el método de stacking, donde en lugar de utilizar votación para combinar, utiliza otro algoritmo de aprendizaje para saber cuáles son los clasificadores más fiables
- Mezcla de clasificadores con un comité aleatorio, es decir, construye distintos clasificadores (de manera aleatoria) y promedia sus predicciones
- Árbol de decisión con clasificadores Naïve Bayes en las hojas (`NBTree`)
- Regresión lineal usando la mediana en lugar de la media (LeastMedSq)

Para cada uno de los clasificadores mencionados, se ha aplicado el algoritmo explicado en la sección anterior, obteniendo el número de aciertos (en la clase fuga sobre el $10\,\%$ de individuos que con mayor probabilidad pertenecen a esa clase) como medida de precisión. Esta medida equivale a la sensibilidad, definida como el cociente entre los verdaderos positivos y la suma de verdaderos positivos y falsos negativos, ya que al medir probabilidades de pertenencia a la clase fuga, no existen ejemplos clasificados como clase no fuga, por lo que el número de verdaderos negativos y falsos positivos son cero. En la tabla 3 se presentan las distintas precisiones de los algoritmos de clasificación anteriores, para un conjunto de entrenamiento formado a partir del $60\,\%$ de elementos de fuga del conjunto original. Están ordenados de mayor a menor precisión media, calculada sobre 10 conjuntos distintos, generados aleatoriamente con los mismos parámetros. Se incluye además la precisión obtenida al evaluar el clasificador sobre el mismo conjunto de entrenamiento, tanto cuando se toma como conjunto el de entrenamiento (sensibilidad máxima con entrenamiento) como el de test (sensibilidad máxima con test), para obtener una idea de cuál sería la máxima precisión alcanzada por el clasificador en cuestión. Esta última medida además nos indica cuánto de sensible es el clasificador a que el conjunto de datos esté desbalanceado.

En la tabla 4 se pueden ver algunas características de los clasificadores mencionados, así como los parámetros utilizados (en Weka) al obtener los resultados recién mencionados.

Como se ha visto anteriormente, la decisión de tomar un $60\,\%$ de clase fuga en el conjunto de entrenamiento no está fundada en ningún resultado previo, por lo que para los mejores métodos de clasificación se muestran en la tabla 5 los resultados con distintos conjuntos de entrenamiento y test con diferentes proporciones del conjunto de fuga original (hay que notar que en esta tabla los resultados mostrados hacen referencia sólo a un conjunto, no a la media obtenida con 10 conjuntos como en la tabla anterior). De la tabla se puede ver que los algoritmos alcanzan su máximo al entrenar con un $70\,\%$ del conjunto de fuga, no obstante, si esta cantidad aumenta, la precisión disminuye, seguramente por sobreajuste.

Como se comentó con anterioridad, aplicar Análisis de Componentes Principales no ha ayudado a la clasificación. Esto se puede ver atendiendo a los resultados de bagging con regresión sin PCA (sensibilidad de $0{,}2775$) frente a cuando se usa PCA (sensibilidad de $0{,}1810$).

También se puede ver la curva de Hit rate frente al tamaño de campaña en la figura 1.

Clasificador	**Sensibilidad media**	**Sensibilidad máxima con entrenamiento**	**Sensibilidad máxima con test**
Bagging con regresión	0,2783	0,7676	0,2742
Regresión lineal	0,2773	0,7594	0,2766
Least Med Sq	0,2772	0,7660	0,0 [error]
$M5'$	0,2756	0,7684	0,2763
Stacking	0,2756	0,7684	0,2763
Bagging con $M5'$	0,2750	0,9296	0,5723
NaiveBayes	0,2418	0,7234	0,2434
NBTree	0,2418	0,7234	0,1563
Comité aleatorio	0,2408	0,9182	0,0
J48	0,2247	0,6997	0,0
J48graft	0,2236	0,6997	0,0
REPTree	0,2124	0,7701	0,2812
Perceptrón multicapa	0,1905	0,8437	0,2561
Bosque aleatorio	0,1820	1,0	1,0
Reglas Part	0,1596	1,0	0,5219
Reglas conjuntivas	0,1554	0,6645	0,1309

Cuadro 3: Sensibilidad de los clasificadores estudiados para un conjunto de entrenamiento formado por el $60\,\%$ de elementos de fuga del conjunto original

Con todos los datos mostrados, podemos elegir el algoritmo de clasificación que se utilizará para clasificar el conjunto de explotación: *bagging con* $M5'$. Se elige este método de clasificación a pesar de que en media no ha sido el mejor que se ha comportado (tabla 3) pero ha demostrado ser robusto frente a cambios de proporción del conjunto de fuga en entrenamiento (tabla 5) y, en particular, a conjuntos de datos no balanceados (cuarta columna de la tabla 3).

Por lo tanto, la lista de individuos que se ha entregado ha sido el $10\,\%$ de las instancias preprocesadas que con mayor probabilidad pertenecen a la clase fuga devueltas por el clasificador bagging con $M5'$, el cual ha sido entrenado con un conjunto de entrenamiento balanceado y con un $70\,\%$ del conjunto de fuga original, ya que es un porcentaje con el cual todos los algoritmos que aparecen en la tabla 5 se comportan adecuadamente.

3. Herramientas utilizadas

Durante el desarrollo de este trabajo se han utilizado dos herramientas:

- Weka, se utilizó la versión 3.5.7 disponible en *www.cs.waikato.ac.nz/ml/weka*. Weka es una colección de algoritmos de aprendizaje automático de código abierto, permite tanto aplicarlos directamente mediante una interfaz gráfica como ser llamados desde código Java. Permite preprocesamiento, clasificación, regresión, clustering, reglas de asociación y visualización de resultados.
- Matlab, se utilizó la versión 7 R14. Es un entorno para el desarrollo de programas de computación numérica, aunque no es de libre distribución, en Internet existen numerosos paquetes que complementan los que vienen por defecto con la aplicación.

Clasificador	**Necesita datos discretos**	**Parámetros**
Bagging con $M5'$	No	`-P 90 -S 1 -I 100 -W weka.classifiers.trees.M5P -- -M 50`
Bagging con regresión	No	`-P 90 -S 1 -I 50 -W weka.classifiers.functions. LinearRegression -- -S 0 -R 0.1`
Bosque aleatorio	Sí	`-I 10 -K 1 -S 1`
Comité aleatorio	Sí	`-S 1 -I 10 -W weka.classifiers.trees. RandomTree -- -K 5 -M 50.0 -S 1`
J48	Sí	`-C 0.0657 -M 50`
J48graft	Sí	`-C 0.15 -M 50`
Least Med Sq	No	`-S 4 -G 0`
$M5'$	No	`-M 50`
NaiveBayes	Sí	(ninguno)
NBTree	Sí	(ninguno)
Perceptrón multicapa	No	`-L 0.3 -M 0.2 -N 150 -V 0 -S 0 -E 20 -H a`
Regla conjuntiva	No	`-N 3 -M 2.0 -P -1 -S 1`
Reglas Part	Sí	`-M 2 -C 0.25 -Q 1`
Regresión lineal	No	`-S 0 -R 0.1`
REPTree	No	`-M 2 -V 0.0010 -N 3 -S 1 -L -1`
Stacking	No	`-X 10 -M "weka.classifiers.functions. LinearRegression -S 0 -R 0.1S 1 -B "weka.classifiers.trees.M5P -M 50.0"`

Cuadro 4: Algunas características de los clasificadores utilizados

Proporción del conjunto de fuga en entrenamiento	Bagging con regresión	Regresión lineal	Least Med Sq	$M5'$	Stacking	Bagging con $M5'$
20 %	0,2773	0,2756	0,2744	0,2761	0,2761	0,2829
30 %	0,2784	0,2786	0,2781	0,2768	0,2769	0,2639
40 %	0,2811	0,2827	0,2793	0,2819	0,2819	0,2771
50 %	0,2787	0,2780	0,2809	0,2802	0,2802	0,2770
60 %	0,2775	0,2751	0,2766	0,2721	0,2721	0,2754
70 %	0,2847	0,2839	0,2855	0,2887	0,2887	0,2799
80 %	0,2722	0,2745	0,2758	0,2734	0,2733	0,2843

Cuadro 5: Sensibilidad sobre distintos conjuntos de entrenamiento y test para los seis mejores algoritmos

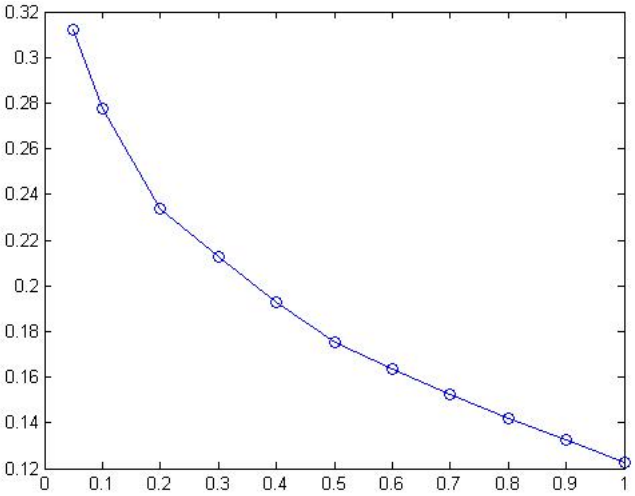

Figura 1: Curva Hit rate frente a tamaño de campaña para bagging con regresión lineal

Cada herramienta se ha utilizado para un propósito específico:

- Para tratar los datos de entrada y realizar gran parte del preprocesamiento se utilizó Matlab, gracias a su facilidad para tratar con matrices y su gran número de utilidades estadísticas. También se utilizó para la creación de los conjuntos de entrenamiento y test, incluyendo las distintas variantes comentadas durante este trabajo. Se intentó utilizar redes neuronales pero por motivos de memoria y tiempo de procesamiento se desestimó.
- Para la parte de clasificadores se utilizó Weka por la facilidad de uso de su interfaz gráfica, que permite probar un algoritmo de clasificación sobre un conjunto de datos en pocos clicks. También se pueden realizar validación cruzada o entrenamiento sobre un conjunto y test sobre otro. A pesar de sus ventajas, para el problema orientado a campaña que nos ocupa se hace necesario programar código Java que acceda a los distintos clasificadores disponibles y realizar operaciones con las predicciones obtenidas.

Los códigos utilizados en las distintas herramientas se encuentran en los apéndices B y C.

4. Conclusiones

En este trabajo se ha abordado un problema con clases desbalanceadas y un gran número de atributos. Para ello, se han utilizado varios métodos de preprocesamiento de datos, pero sobre todo, se han analizado numeros algoritmos de clasificación, destacando aquellos que utilizan regresión lineal en algún momento ($M5'$, regresión lineal, Least Med Sq, bagging o stacking con alguno de los anteriores). El método elegido ha sido uno de los que mejores resultados ha dado durante las pruebas y el que ha demostrado ser más robusto frente a conjuntos de test desbalanceados: bagging con $M5'$.

Para terminar, hay que comentar que, ya que el método elegido ha sido de tipo bagging, estos se pueden mejorar bastante utilizando un ordenamiento especial al agregar los clasificadores [5]. Esta observación, junto con un análisis en detalle de la influencia de la creación de los conjuntos de entrenamiento y prueba (ver sección 2.2) (en particular, de la forma de elegir los casos positivos y negativos,

además del número de casos de fuga a considerar), podría formar parte del trabajo futuro con respecto a lo aquí expuesto de cara a mejorar la predicción buscada.

A. Lista de atributos definitivos

- CLASE
- ANT_MUT
- ANT_POL
- AGENTE
- CANAL_P
- GRUPO
- PROPIET
- USO_VEH
- FP
- FECHA_COM
- PROCED
- TIPO_ANT
- FECHA_CARNET
- CP_POL
- TERRITORIAL
- ZONA
- SEXO
- FNAC_P
- COD_EDAD
- TIPO
- BONF
- VIAJE
- CAR
- TIPO_VEH
- POTE
- PRECIO
- AFE2

- AFE3
- IMPORT
- CANAL_M
- CP_M
- ECIV
- FECHA_NACM
- PUBLI
- nramos
- PVPh15000
- PVP15000_25000
- PVPmas25000
- recibosdev
- prima
- prima_media
- prima_total

B. Código Matlab para crear los conjuntos de entrenamiento y test

```
%Leer datos
construccion = dlmread('construccion_matlab.txt', ';');
%Añadir titulos
titulos = strvcat('ID_NUMERICO', 'CLASE', 'PROX_REC', 'ANT_MUT', '
    ANT_POL', 'AGENTE', 'CANAL_P', 'GRUPO', 'PROPIET', 'COND', '
    AUTOESCUELA', 'ALQUILER', 'USO_VEH', 'FP', 'FECHA_COM', 'PROCED',
    'YCIA_ANT', 'TIPO_ANT', 'FECHA_CARNET', 'CP_POL', 'TERRITORIAL', '
    ZONA', 'SEXO', 'FNAC_P', 'COD_EDAD', 'TIPO', 'BONF', 'VIDA', '
    VIAJE', 'CAR', 'DEF', 'TIPO_VEH', 'POTE', 'PRECIO', 'FCV1', 'FSN1'
    , 'FSN2', 'FSN3', 'AFE1', 'AFE2', 'AFE3', 'IMPORT', 'FIM1', 'FIM2'
    , 'FIM3', 'FAS1', 'FAS2', 'CANAL_M', 'CLUB', 'CP_M', 'ECIV', '
    FECHA_NACM', 'PUBLI', 'CA', 'MES', 'AGNO', 'pclubp', 'publici', '
    activ_hogar', 'activ_accid', 'activ_comer', 'activ_rc', 'nramos',
    'PVPh15000', 'PVP15000_25000', 'PVPmas25000', 'recibosdev', 'prima
    ', 'prima_media', 'prima_total');
%Eliminamos la última columna: en el fichero hay datos ahí
construccion = construccion(:, 1:70);
%El primer indice (17) tiene un 22% de missing values
%Eliminamos el atributo 17 = ycia_ant
%El primer atributo (id: 1) no interesa
%Los atributos prox_rec (3) y agno (56) nunca varian
```

```
%El atributo mes (55) varia muy poco y no interesa
%Los atributos 53 y 58 son iguales (publici): quitar 58
%Los siguientes atributos contienen muchos valores raros (999)
%aunque en principio son útiles:
%35-38, 43-47
%Los atributos autoescuela y alquiler son casi constantes: 11, 12
%Conclusión: quitar
%1, 3, 10, 11, 12, 17, 28, 31, 35, 36, 37, 38, 39,
%43, 44, 45, 46, 47, 49, 54, 55, 56, 57, 58, 59, 60, 61, 62
datos = [construccion(:, 2), construccion(:, 4:9), construccion(:,
    13:16), construccion(:, 18:27), ...
     construccion(:, 29:30), construccion(:, 32:34), construccion(:,
         40:42), construccion(:, 48), construccion(:, 50:53),
         construccion(:, 63:70)];
datosTitulos = strvcat(titulos(2,:), titulos(4:9,:), titulos(13:16,:)
    , titulos(18:27,:), ...
     titulos(29:30,:), titulos(32:34,:), titulos(40:42,:), titulos
         (48,:), titulos(50:53,:), titulos(63:70,:));
[rows, cols] = size(datos);
%0.1 Sustituir missing values por su media
for (i=1:cols),
     indicesNoMissing = find(~isnan(datos(:,i)));
     media = mean(datos(indicesNoMissing, i));
     datos(find(isnan(datos(:,i))), i) = media;
end
%0.2 Centramos y normalizamos (la clase no)
avg = mean(datos(:,2:cols));
desv = std(datos(:,2:cols));
avgM = ones(rows,1)*avg;
desvM = diag(desv);
datosNorm = [datos(:,1), (datos(:,2:cols)-avgM)/desvM];
%0.3. Eliminar instancias con outliers
%0.3.1. Encontrar atributos con outliers
numInstancias = size(datosNorm,1);
mu = mean(datosNorm);
sigma = std(datosNorm);
outliers = abs(datosNorm - mu(ones(numInstancias,1),:)) > 3.8*sigma(
    ones(numInstancias,1),:);
%    En nout encontramos el numero de outliers por atributo
nout = sum(outliers);
datosNoOutliers = [];
for (i=1:numInstancias),
     if isempty(find(outliers(i,:)>0))
          datosNoOutliers = [datosNoOutliers; datosNorm(i,:)];
     end
end

s = readStringMatrix(datosTitulos, ',');

fid = fopen(['datosNoOutliers.csv'], 'w');
```

```
fprintf(fid, s);
fprintf(fid, '\n');
fclose(fid);
dlmwrite(['datosNoOutliers.csv'], datosNoOutliers, '-append');

%Elegir conjunto para aprendizaje
%1. Calcular numero de clase minoritaria
fuga = find(datosNoOutliers(:,1)==1);
[rowsFuga, colsFuga] = size(fuga);
%2. Escoger rows instancias de la clase mayoritaria al azar
nofuga = find(datosNoOutliers(:,1)==0);
[rowsNoFuga, colsNoFuga] = size(nofuga);
%    Mezclamos la clase mayoritaria
nofugaAleat = nofuga(randperm(rowsNoFuga));

%Proporción a mantener para el conjunto de test:
propFuga = rowsFuga/(rowsNoFuga+rowsFuga);
propNoFuga = 1-propFuga;

PROP_TEST = 0.4;
PROP_TRAIN = 1-PROP_TEST;
maxRowsFugaTest = floor(PROP_TEST*rowsFuga);
maxRowsFugaTrain = floor(PROP_TRAIN*rowsFuga);

fugaTest = fuga(1:maxRowsFugaTest,:);
fugaTrain = fuga((maxRowsFugaTest+1):(maxRowsFugaTrain+
    maxRowsFugaTest),:);
noFugaAddTest = floor(maxRowsFugaTest*propNoFuga/propFuga);
maxRowsNoFugaTest = maxRowsFugaTest + noFugaAddTest;
nofugaACombinarTest = nofugaAleat(1:maxRowsNoFugaTest,:);
minRowsNoFugaTrain = maxRowsNoFugaTest+1;
maxRowsNoFugaTrain = maxRowsFugaTrain+maxRowsNoFugaTest;
nofugaACombinarTrain = nofugaAleat(minRowsNoFugaTrain:
    maxRowsNoFugaTrain,:);

%3. Crear conjuntos de entrenamiento y test
datosCombinadosTrain = [datosNoOutliers(fugaTrain,:); datosNoOutliers
    (nofugaACombinarTrain, :)];
numInstancias = size(datosCombinadosTrain,1);
indices = randperm(numInstancias);
datosCombinadosTrain = datosCombinadosTrain(indices, :);

datosCombinadosTest = [datosNoOutliers(fugaTest,:); datosNoOutliers(
    nofugaACombinarTest, :)];
numInstancias = size(datosCombinadosTest,1);
indices = randperm(numInstancias);
datosCombinadosTest = datosCombinadosTest(indices, :);

fid = fopen(['datosCombinadosTestNoOutliers.csv'], 'w');
fprintf(fid, s);
```

```
fprintf(fid, '\n');
fclose(fid);
dlmwrite(['datosCombinadosTestNoOutliers.csv'], datosCombinadosTest,
    '-append');
fid = fopen(['datosCombinadosTrainNoOutliers.csv'], 'w');
fprintf(fid, s);
fprintf(fid, '\n');
fclose(fid);
dlmwrite(['datosCombinadosTrainNoOutliers.csv'], datosCombinadosTrain
    , '-append');
```

C. Código Java para evaluar un clasificador según una campaña del 10 %

```
private static double doClassification(Classifier classifier,
    Instances insTest) throws Exception {
    TreeMap<Double, ArrayList<Integer>> distributionInstances = new
        TreeMap<Double, ArrayList<Integer>>();
    int trueNegative = 0;
    int falseNegative = 0;
    int falsePositive = 0;
    int truePositive = 0;
    for (int i = 0; i < insTest.numInstances(); i++) {
        Instance ins = insTest.instance(i);
        double value = classifier.classifyInstance(ins);
        double real = ins.classValue();
        if (real != 0.0) {
            // clase 1
            if (value != 0.0) {
                truePositive++;
            } else {
                falsePositive++;
            }
        } else {
            // clase 0
            if (value != 0.0) {
                falseNegative++;
            } else {
                trueNegative++;
            }
        }
        double[] p = classifier.distributionForInstance(ins);
        double distributionClassFuga = p[p.length - 1];
        ArrayList<Integer> indices = new ArrayList<Integer>();
        if (distributionInstances.containsKey(distributionClassFuga))
            {
```

```
            indices = distributionInstances.get(distributionClassFuga
                );
        }
        indices.add(i);
        distributionInstances.put(distributionClassFuga, indices);
    }
    double total = 1.0 * (trueNegative + falseNegative +
        falsePositive + truePositive);
    System.out.println("Era a y dice a =" + trueNegative + " (" + ((
        double) trueNegative / total) + " %)");
    System.out.println("Era a y dice b =" + falseNegative + " (" + ((
        double) falseNegative / total) + " %)");
    System.out.println("Era b y dice a =" + falsePositive + " (" + ((
        double) falsePositive / total) + " %)");
    System.out.println("Era b y dice b =" + truePositive + " (" + ((
        double) truePositive / total) + " %)");
    // Evaluation of 10%
    Instances instancesFuga = new Instances(insTest, 0);
    long max = Math.round(0.1 * total);
    int n = 0;
    ArrayList<Integer> indexInstancesFuga = new ArrayList<Integer>();
    Iterator<Double> it = distributionInstances.descendingKeySet().
        iterator();
    for (; it.hasNext();) {
        double key = it.next();
        ArrayList<Integer> ins = distributionInstances.get(key);
        int i = 0;
        while ((n < max) && (i < ins.size())) {
            indexInstancesFuga.add(ins.get(i));
            instancesFuga.add(insTest.instance(ins.get(i)));
            i++;
            n++;
        }
        if (n == max) {
            break;
        }
    }
    System.out.println("Evaluando " + max + " instancias de " + total
        );
    trueNegative = 0;
    falseNegative = 0;
    falsePositive = 0;
    truePositive = 0;
    Evaluation eval = new Evaluation(instancesFuga);
    eval.evaluateModel(classifier, instancesFuga);
    for (int i = 0; i < indexInstancesFuga.size(); i++) {
        int index = indexInstancesFuga.get(i);
        double real = insTest.instance(index).classValue();
        double value = classifier.classifyInstance(insTest.instance(
            index));
```

```
            if (real != 0.0) {
                // clase 1
                if (value != 0.0) {
                    truePositive++;
                } else {
                    falsePositive++;
                }
            } else {
                // clase 0
                if (value != 0.0) {
                    falseNegative++;
                } else {
                    trueNegative++;
                }
            }
        }
        System.out.println("Era a y dice a =" + trueNegative + " (" + ((
            double) trueNegative / max) + " %)");
        System.out.println("Era b y dice a =" + falsePositive + " (" + ((
            double) falsePositive / max) + " %)");
        System.out.println("Era a y dice b =" + falseNegative + " (" + ((
            double) falseNegative / max) + " %)");
        System.out.println("Era b y dice b =" + truePositive + " (" + ((
            double) truePositive / max) + " %)");

        double sens = (double) truePositive / ((double) (truePositive +
            falseNegative));
        double spec = (double) trueNegative / ((double) (trueNegative +
            falsePositive));
        double acc = ((double) (trueNegative + falsePositive)) / max;
        System.out.println("Sensitivity = " + sens);
        System.out.println("Specificity = " + spec);
        System.out.println("Accuracy = " + acc);
        System.out.println("ROC point [FPR, TPR] = " + (1 - spec) + ", "
            + sens);
        return ((double) truePositive / max);
    }
```

Referencias

[1] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[2] Iván Cantador. Aplicación de perceptrones paralelos y adaboost a problemas de clasificación desequilibrados. Master's thesis, Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid (UAM). Madrid, España, Junio 2005.

[3] Laurie Davies and Ursula Gather. The identification of multiple outliers. *Journal of the American Statistical Association*, 88(423):782–792, 1993.

[4] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, Noviembre 2000.

[5] Gonzalo Martínez-Muñoz and Alberto Suárez. Aggregation ordering in bagging. In *Proc. of the IASTED International Conference on Artificial Intelligence and Applications*, pages 258–263. Acta Press, 2004.

[6] G. Weiss and F. Provost. The effect of class distribution on classifier learning. Technical report, 2001.

[7] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, Junio 2005.